



# Алгоритмы и структуры данных

## Лекция 6. Продвинутое методы сортировки

---

Антон Штанюк (к.т.н, доцент)

22 марта 2021 г.

Нижегородский государственный технический университет им. Р.Е. Алексеева  
Институт радиоэлектроники информационных технологий  
Кафедра "Компьютерные технологии в проектировании и производстве"

Быстрая сортировка

Сортировка Шелла

Сортировка слиянием

Список литературы

В этой теме мы рассмотрим некоторые популярные алгоритмы сортировки, имеющие более выгодные показатели временной сложности, чем простые методы, рассмотренные ранее.

Мы рассмотрим:

1. Быструю сортировку (Хоара).
2. Сортировку Шелла.
3. Сортировку слиянием.

## Быстрая сортировка

---

**QuickSort** - широко известный алгоритм сортировки, разработанный английским учёным Чарльзом Хоаром. Один из быстрых известных универсальных алгоритмов сортировки массивов, являющийся также, наиболее распространённым.

В стандартную библиотеку C/C++ входит функция **qsort**. Краткое описание работы:

- выбрать элемент, называемый опорным
- сравнить все остальные элементы с опорным, на основании сравнения разбить множество на три — "меньшие опорного", "равные" и "большие", расположить их в порядке меньшие-равные-большие.
- повторить рекурсивно для "меньших" и "больших".

```
void group(int *arr, int size, int value) {  
    int left=0, right=size-1;  
    while (left<right) {  
        if(arr[left]<=value)  
            left++;  
        else if(arr[right]>value)  
            right--;  
        else if(arr[left]>value && arr[right]<=value) {  
            int t=arr[left];  
            arr[left]=arr[right];  
            arr[right]=t;  
        }  
    }  
}
```

Данный алгоритм демонстрирует распространенный прием, когда обработка массива идет с двух концов навстречу друг другу.

Например, если массив содержал следующие элементы:

4 8 9 2 5 7 9 1 3 7

То после группировки относительно значения 5 он превратится:

4 3 1 2 5 7 9 9 8 7

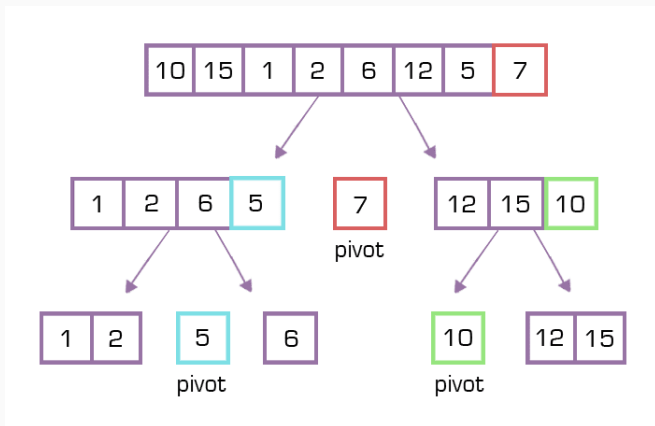
Алгоритм быстрой сортировки является одним из самых быстрых: среднее время работы близко к  $O(n \log n)$ . Кроме того, он не требует дополнительной памяти, за исключением расходов на стек вызовов.

Быстрая сортировка использует рекурсию. Рассмотрим работу алгоритма применительно к участку массива  $arr[p..r]$ .

- Элементы массива переставляются так, чтобы любой из элементов  $arr[p], \dots, arr[q]$  был не больше любого из элементов  $arr[q+1], \dots, arr[r]$ , где  $q$  - некоторое число в интервале  $p \leq q < r$ . Эта операция называется разделением (partition). - Процедура сортировки рекурсивно вызывается для массивов  $arr[p..q]$  и  $arr[q+1..r]$ . После этого исходный массив  $arr[p..r]$  отсортирован.



Пример сортировки с выбором опорного элемента из конца массива



# Быстрая сортировка

```
void QuickSort(int arr[], int p, int r)
{
    int i,j;
    int x;
    i=p;
    j=r;
    x=arr[(i+j)/2];
    do
    {
        while(arr[i]<x) i++;
        while(arr[j]>x) j--;
        if(i<=j) {
            swap(&arr[i],&arr[j]); // функция обмена местами двух элтов—
            i++;
            j--;
        }
    }
    while(i<=j);
    if(j>p)
        QuickSort(arr,p,j);
    if(i<r)
        QuickSort(arr,i,r);
}
```

Достоинства:

- Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения.
- Прост в реализации.
- Требуется лишь  $O(\log n)$  дополнительной памяти для своей работы.
- Хорошо сочетается с механизмами кэширования и виртуальной памяти.

## Недостатки:

- Сильно деградирует по скорости (до  $O(n^2)$ ) при неудачных выборах опорных элементов, что может случиться при неудачных входных данных. Этого можно избежать, выбирая опорный элемент случайно, а не фиксированным образом.
- Наивная реализация алгоритма может привести к ошибке переполнения стека, так как ей может потребоваться сделать  $O(n)$  вложенных рекурсивных вызовов. В улучшенных реализациях, в которых рекурсивный вызов происходит только для сортировки большей из двух частей массива, глубина рекурсии гарантированно не превысит  $O(\log n)$ .

## Сортировка Шелла

---

Идея метода заключается в сравнение разделенных на группы элементов последовательности, находящихся друг от друга на некотором расстоянии. Изначально это расстояние равно  $d$  или  $N/2$ , где  $N$  — общее число элементов. На первом шаге каждая группа включает в себя два элемента расположенных друг от друга на расстоянии  $N/2$ ; они сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние  $d$  сокращается на  $d/2$ , и количество групп, соответственно, уменьшается. Постепенно расстояние между элементами уменьшается, и на  $d = 1$  проход по массиву происходит в последний раз.

Исходный  
массив

5 3 8 0 7 4 9 1 6 2

d=5

4 3 1 0 2 5 9 8 6 7

d=2

1 0 2 3 4 5 6 7 9 8

d=1

0 1 2 3 4 5 6 7 8 9

```
void ShellSort(int arr[], int size)
{
    int j, d, temp;
    d=size/2;
    while (d>0)
    {
        for (int i=0; i<size-d; i++)
        {
            j=i;
            while (j>=0 && arr[j]>arr[j+d])
            {
                temp=arr[j];
                arr[j]=arr[j+d];
                arr[j+d]=temp;
                j--;
            }
        }
        d=d/2;
    }
}
```



Интересно, но качество данного алгоритма зависит от последовательности значений  $d$ .

Существует несколько подходов к выбору этих значений:

1. Первоначально используемая Шеллом последовательность длин промежутков:  
 $d_1 = N/2, d_i = d_{i-1}/2, d_k = 1$  в худшем случае, сложность алгоритма составит  $O(N^2)$
2. Предложенная Хиббардом последовательность: все значения  $2^i - 1 \leq N, i \in \mathbb{N}$  такая последовательность шагов приводит к алгоритму сложностью  $O(N^{3/2})$
3. Предложенная Седжвиком последовательность:  
 $d_i = 9 \cdot 2^i - 9 \cdot 2^{i/2} + 1$ , если  $i$  четное и  
 $d_i = 8 \cdot 2^i - 6 \cdot 2^{(i+1)/2} + 1$ , если  $i$  нечетное.  
При использовании таких приращений средняя сложность алгоритма составляет:  $O(n^{7/6})$ , а в худшем случае порядка  $O(n^{4/3})$ .

## Сортировка слиянием

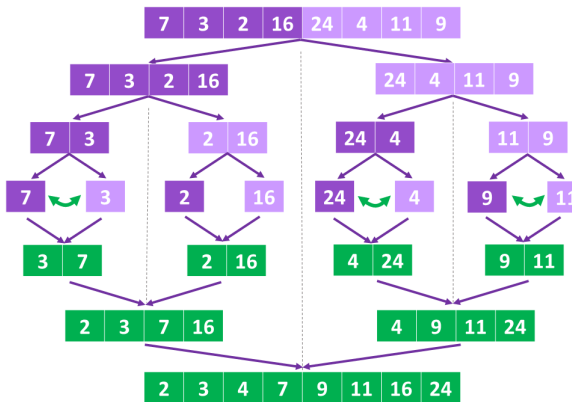
---

**Слияние** означает объединение двух (или более) последовательностей в одну упорядоченную последовательность при помощи циклического выбора элементов, доступных в данный момент.

Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Затем их решения комбинируются, и получается решение исходной задачи.

Процедура слияния предполагает объединение двух предварительно упорядоченных подпоследовательностей размерности  $n/2$  в единую последовательность размерности  $n$ . Начальные элементы предварительно упорядоченных последовательностей сравниваются между собой, и из них выбирается наименьший. Соответствующий указатель перемещается на следующий элемент. Процедура повторяется до тех пор, пока не достигнут конец одной из подпоследовательностей. Оставшиеся элементы другой подпоследовательности при этом передаются в результирующую последовательность в неизменном виде.

## Merge Sort



**Step 1:**  
Split sub-lists in two until you reach pair of values.

**Step 3:**  
Sort/swap pair of values if needed.

**Step 4:**  
Merge and sort sub-lists and repeat process till you merge to the full list.

```
int* merge_sort(int *up, int *down, unsigned int left, unsigned int right)
{
    if (left == right)
    {
        down[left] = up[left];
        return down;
    }

    unsigned int middle = (left + right) / 2;

    // разделяй и властвуй
    int *lbuf = merge_sort(up, down, left, middle);
    int *rbuf = merge_sort(up, down, middle + 1, right);

    // слияние двух отсортированных половин
    int *target = lbuf == up ? down : up;
    ...
}
```

```
...
unsigned int lcur = left, rcur = middle + 1;
for (unsigned int i = left; i <= right; i++)
{
    if (lcur <= middle && rcur <= right)
    {
        if (lbuf[lcur] < rbuf[rcur])
        {
            target[i] = lbuf[lcur];
            lcur++;
        }
        else
        {
            target[i] = rbuf[rcur];
            rcur++;
        }
    }
}
...
```

```
...  
else if (lcur <= middle)  
{  
    target[i] = lbuf[lcur];  
    lcur++;  
}  
else  
{  
    target[i] = rbuf[rcur];  
    rcur++;  
}  
}  
return target;  
}
```



```
int *arr1=new int[N];
int *arr2=new int[N];






for(int i=0;i<N;i++)
    arr1[i]=rand()%100;






merge_sort(arr1,arr2,0,N-1);

for(int i=0;i<N;i++)
    cout<<arr2[i]<<endl;
```

## Список литературы

---

-  Кормен Т.,Лейзерсон Ч., Ривест Р.  
*Алгоритмы: построение и анализ*  
МЦНМО, Москва, 2000
-  Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы:  
построение и анализ.  
*2-е изд. — М.: «Вильямс», 2006*
-  Википедия  
*Алгоритм*  
<http://ru.wikipedia.org/wiki/Алгоритм>
-  Википедия  
*Список алгоритмов*  
[http://ru.wikipedia.org/wiki/Список\\_алгоритмов](http://ru.wikipedia.org/wiki/Список_алгоритмов)
-  Традиция  
*Задача коммивояжёра*  
<http://traditio.ru/wiki/Задача>

-  Википедия  
*NP-полная задача*  
<http://ru.wikipedia.org/wiki/NP-полная>
-  Серджвик Р.  
*Фундаментальные алгоритмы на C++. Части 1-4*  
Diasoft, 2001
-  Седжвик Р.  
*Фундаментальные алгоритмы на C. Анализ/Структуры данных/Сортировка/Поиск*  
СПб.: ДиаСофтЮП, 2003
-  Седжвик Р.  
*Фундаментальные алгоритмы на C. Алгоритмы на графах*  
СПб.: ДиаСофтЮП, 2003
-  Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы.  
*Издательский дом «Вильямс», 2000*



Кнут Д.

*Искусство программирования, том 1. Основные алгоритмы*  
3-е изд. — М.: «Вильямс», 2006



Кнут Д.

*Искусство программирования, том 2. Получисленные методы*  
3-е изд. — М.: «Вильямс», 2007



Кнут Д.

*Искусство программирования, том 3. Сортировка и поиск*  
2-е изд. — М.: «Вильямс», 2007



Кнут Д.

*Искусство программирования, том 4, выпуск 3. Генерация всех сочетаний и разбиений*  
М.: «Вильямс», 2007



Кнут Д.

*Искусство программирования, том 4, выпуск 4. Генерация всех деревьев. История комбинаторной генерации*

М.: «Вильямс», 2007